DOCUMENTAÇÃO DO PROJETO: EVENTHIVE

Descrição do Projeto

O EventHive é uma plataforma web completa de gestão e venda de bilhetes para eventos, desenvolvida como projeto final da formação Web Development. O sistema integra tecnologias modernas (PHP, MySQL, React) para criar uma solução robusta e escalável.

OBJETIVO PRINCIPAL

Criar um sistema de gestão de eventos que permita aos utilizadores visualizar, pesquisar e comprar bilhetes para eventos, enquanto oferece aos administradores ferramentas completas de gestão.

FUNCIONALIDADES CORE

- Gestão de Utilizadores: Registo, autenticação, perfis e histórico
- Gestão de Eventos: CRUD completo com categorias, múltiplos tipos de bilhete e upload de imagens
- Sistema de Compras: Carrinho de compras, checkout seguro e controlo automático de stock
- Painel Administrativo: Dashboard com estatísticas, CRUDs completos e sistema de logs
- Interface Responsiva: Design Bootstrap adaptável a todos os dispositivos

TECNOLOGIAS IMPLEMENTADAS

- Backend: PHP 8 com arquitetura RESTful, MySQL com triggers e views
- Frontend: React 19+ com Context API, React Router e componentes reutilizáveis
- Segurança: Sessões seguras, prepared statements, validação de dados e auditoria
- Base de Dados: Estrutura normalizada com integridade referencial e automação~

ARQUITETURA

Sistema modular com separação clara entre frontend (SPA React) e backend (API REST PHP), garantindo escalabilidade, manutenibilidade e reutilização de código.

1. PREPARAÇÃO DO AMBIENTE DE TRABALHO (resumo)

1.1 Instalação do Composer (PHP)

- Download: https://getcomposer.org/
- Instrução: Instalar globalmente e validar no terminal com o comando composer --version.

1.2 Instalação do XAMPP

- Download: https://www.apachefriends.org/
- Instruções: Instalar e iniciar os módulos Apache e MySQL. O projeto deve ser colocado no diretório C:/xampp/htdocs/EventHive/.

1.3 Instalação do MySQL Workbench

- Download: https://dev.mysql.com/downloads/workbench/
- Instruções: Ligar à base de dados local (localhost/root) e importar o script my_script_completed.sql.

1.4 Instalação do React

- Pré-requisito: Instalar Node.js e npm a partir de: https://nodejs.org/
- Instruções:
    1. Criar o projeto frontend com npx create-react-app frontend.
    2. Instalar as dependências necessárias executando npm install no diretório frontend.

1.5 Instalação de Dependências

- PHP (via Composer): vlucas/phpdotenv, PDO (extensão nativa).
- React (via npm): axios, react-router-dom, styled-components, bootstrap, etc.

2. ESTRUTURA DE PASTAS E ARQUITETURA

Estrutura das pastas e ficheiros Backend:

api/

admin/

categories_crud.php

dashboard_stats.php

events_crud.php

orders_list.php

users_crud.php

auth/

change_password.php

login.php

logout.php

profile.php

register.php

events/

get_all_events.php

get_event_details.php

get_event_by_token.php

get_event_by_category.php

categories/

get_all_categories.php

cart/

add_to_cart.php

checkout.php

get_cart.php

remove_from_cart.php

user/

purchase_history.php

config/

Database.php

bootstrap.php

core/

models/

Category.php

Event.php

Order.php

User.php

classes/

Auth.php

SessionManager.php

lib/

uploads/

vendor/

Estrutura das pastas e ficheiros - Frontend

frontend/

.env

.gitignore

package.json

package-lock.json

README.md

public/

android-chrome-192x192.png

android-chrome-512x512.png

apple-chrome-512x512.png

apple-touch-icon.png

favicon-16x16.png

favicon-32x32.png

favicon.ico

index.html

logo.png

logo192.png

logo512.png

manifest.json

src/

App.js

index.js

index.css

reportWebVitals.js

assets/

images/

events_hero.png

logo-nobg1.png

utils/

constantes.js

components/

events/

EventCard.jsx

layout/

Footer.jsx

Navbar.jsx

UI/

Button.jsx

Message.jsx

ReusableTable.jsx

Spinner.jsx

context/

AuthContext.js

AuthProvider.js

CartContext.js

CartProvider.js

pages/

AdminDashboardPage.jsx

CartPage.jsx

categories_crud_page.jsx

EventDetailPage.jsx

EventsPage.jsx

event_crud_page.jsx

HomePage.jsx

list_orders_page.jsx

LoginPage.jsx

NotFoundPage.jsx

ProfilePage.jsx

RegisterPage.jsx

users_crud_page.jsx

services/

api.js

authService.js

eventService.js

categoryService.js

ordersService.js

Estrutura das pastas e ficheiros databases/

backups/

backup_eventhive_db.sql

diagrama_ER.JPG

my_script_completed.sql

Pasta documentação/

documentaçao_HeventHive.pdf

testes_HeventHive.pdf

preparacao_de_ambiente_de_trabalho_HeventHive.pdf

3. BASE DE DADOS MYSQL E DIAGRAMA ER

3.1 Estrutura e Scripts

O script principal my_script_completed.sql executa as seguintes ações:

- Criação da base de dados eventhive_db.
- Criação das tabelas: roles, users, categories, events, tickets, orders, order_items, e activity_logs.
- Definição de chaves primárias, estrangeiras e restrições.
- Criação da View vw_event_details para simplificar consultas.
- Implementação do Trigger trg_after_order_item_insert para controlo de stock.
- Inserção de dados de teste para popular o sistema.

3.2 Estrutura das Tabelas

- roles: Define as permissões (id_role, role_name).
- users: Armazena os dados dos utilizadores.
- categories: Categorias para os eventos.
- events: Informação principal dos eventos.
- tickets: Tipos de bilhete por evento, com preço e quantidade.
- orders: Registo das compras efetuadas.
- order_items: Detalhes dos bilhetes em cada compra.
- activity_logs: Registo de ações importantes no sistema.

3.3 Trigger de Controlo de Stock de Bilhetes

O trigger trg_after_order_item_insert atualiza automaticamente a quantidade de bilhetes disponíveis após cada inserção na tabela order_items.

```
CREATE TRIGGER trg_after_order_item_insert
AFTER INSERT ON order_items
FOR EACH ROW
BEGIN
   UPDATE tickets
   SET quantity_available = quantity_available - NEW.quantity
   WHERE id_ticket = NEW.id_ticket;
END;
```

3.4 View para Detalhes de Eventos

A view vw_event_details junta dados das tabelas events e categories para facilitar a listagem de eventos.

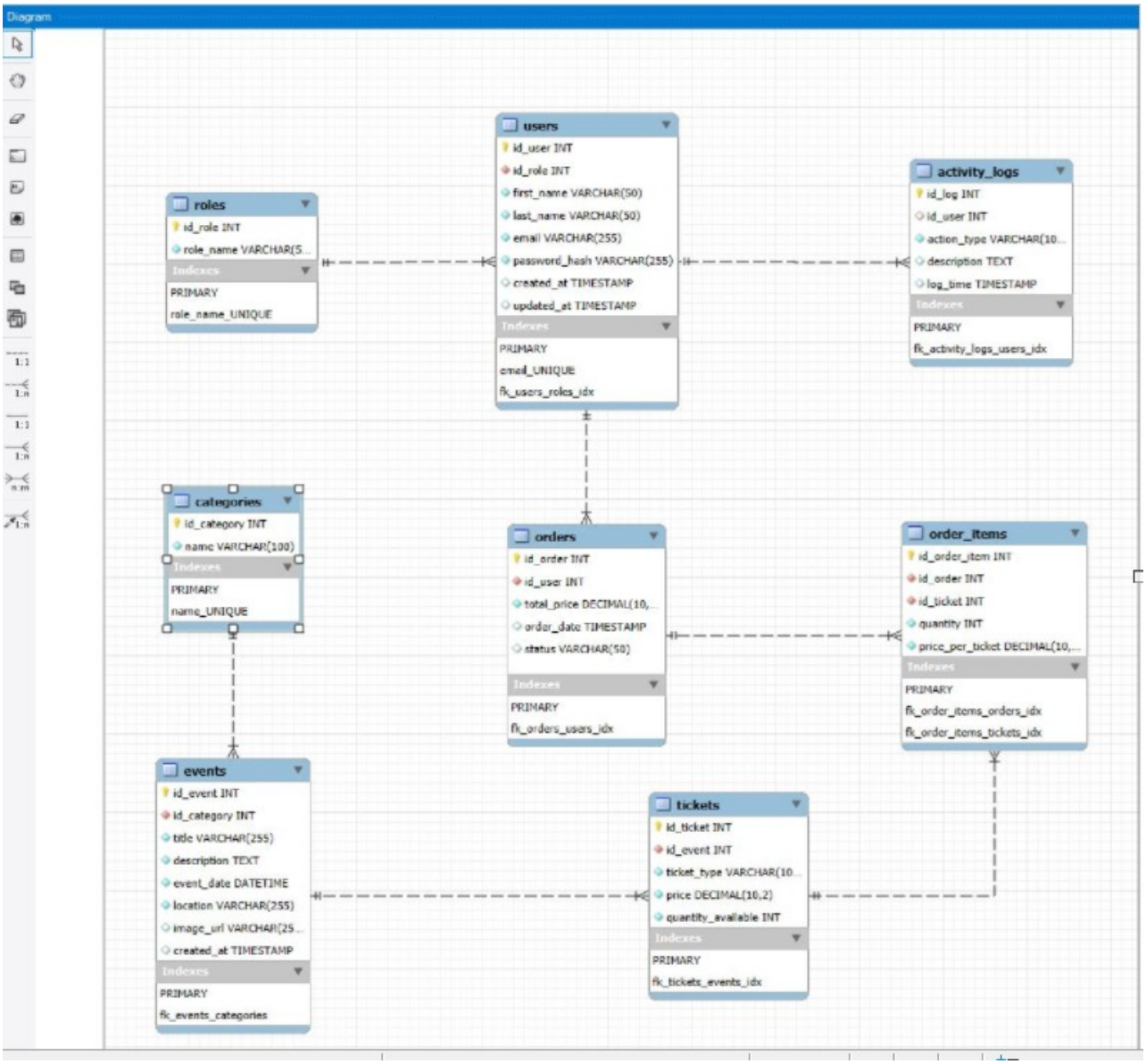
```
CREATE OR REPLACE VIEW vw_event_details AS
SELECT
   e.id_event, e.title, e.description, e.event_date,
   e.location, e.image_url, c.name AS category_name
FROM events AS e
JOIN categories AS c ON e.id_category = c.id_category;
```

3.5 Dados de Teste Essenciais (ver my_script_completed.sql na pasta Database para inserir dados de teste)

- User Teste Administrador: admin@eventhive.com / Adminpass1.
- User Teste Cliente: cliente@eventhive.com / Adminpass1.
- Foram inseridos 7 eventos, 10 tipos de bilhetes, e compras de exemplo.

3.6 Diagrama ER e Relações

Ficheiro do Diagrama:



- Relações entre Tabelas:
    - users → roles: Chave Estrangeira id_role (Relação 1:N - Um role pode ter muitos utilizadores).
    - events → categories: Chave Estrangeira id_category (Relação 1:N - Uma categoria pode ter muitos eventos).
    - tickets → events: Chave Estrangeira id_event (Relação 1:N - Um evento pode ter muitos tipos de bilhete).
    - orders → users: Chave Estrangeira id_user (Relação 1:N - Um utilizador pode ter muitas encomendas).
    - order_items → orders: Chave Estrangeira id_order (Relação 1:N - Uma encomenda pode ter muitos itens).
    - order_items → tickets: Chave Estrangeira id_ticket (Relação 1:N - Um tipo de bilhete pode estar em muitos itens de encomenda).
    - activity_logs → users: Chave Estrangeira id_user (Relação 1:N - Um utilizador pode ter muitos logs de atividade).

- Integridade Referencial:
    - Todas as chaves estrangeiras foram implementadas com restrições (CASCADE/RESTRICT) para garantir a consistência dos dados.
    - Existe um trigger automático para manter o stock de bilhetes atualizado.
    - Foram aplicadas validações de unicidade a campos críticos (ex: email do utilizador, nomes de categorias).

3.7 Backups

- Local: databases/backups/
- Comando para Backup: mysqldump -u root -p eventhive_db > "databases/backups/backup_eventhive_db.sql";
- Comando para Restauro: mysql -u root -p eventhive_db < "databases/backups/backup_eventhive_db.sql";

4. BACKEND (API RESTful em PHP)

4.1 Estrutura do Backend

- api/: Endpoints RESTful, organizados por recurso.
- config/: Configuração da base de dados e bootstrap.
- core/: Modelos de dados (User, Event) e classes utilitárias (Auth, SessionManager).
- uploads/: Armazenamento de imagens dos eventos.
- vendor/: Dependências geridas pelo Composer.

4.2 Endpoints da API

Administração (/api/admin/)

- categories_crud.php, dashboard_stats.php, events_crud.php, orders_list.php, users_crud.php.

Autenticação (/api/auth/)

- register.php, login.php, logout.php, profile.php, change_password.php.

Eventos (/api/events/)

- get_all_events.php, get_event_details.php, get_event_by_token.php, get_event_by_category.php

Categorias(/api/categories)

- get_all_categories.php

Carrinho e Checkout (/api/cart/)

- add_to_cart.php, get_cart.php, remove_from_cart.php, checkout.php.

Perfil de Utilizador (/api/user/)

- purchase_history.php.

4.3 Exemplos de Requisições

Login: POST /api/auth/login.php

- Body: { "email": "admin@eventhive.com", "password": "Adminpass1." }
- Resposta: { "success": true, "message": "Login efetuado com sucesso." }

Adicionar ao Carrinho: POST /api/cart/add_to_cart.php

- Body: { "id_ticket": 1, "quantity": 2 }
- Resposta: { "success": true, "message": "Produto adicionado ao carrinho." }

Estatísticas (Admin): GET /api/admin/dashboard_stats.php

- Resposta: { "success": true, "stats": {"total_events": 7, ...} }

4.4 Lógica, Segurança e Boas Práticas

- Uso de Prepared Statements em todas as queries para prevenir SQL Injection.
- Gestão de sessão segura com regeneração de ID.
- Proteção de endpoints de administração com base na role do utilizador.
- Respostas da API em formato JSON padronizado.
- Validação rigorosa de todos os dados de entrada.

5. FRONTEND (REACT)

5.1 Estrutura Detalhada do Frontend

- src/assets/: Imagens, e outros ficheiros estáticos.
- src/components/: Componentes reutilizáveis (ex: EventCard.jsx, Button.jsx, Navbar.jsx).
- src/context/: Gestão de estado global com Context API (AuthContext.js, CartContext.js e etc).
- src/pages/: Componentes que representam cada página da aplicação (HomePage.jsx, AdminDashboardPage.jsx e etc...).
- src/services/: Funções para comunicar com a API do backend (authService.js,
- eventService.js).
-

5.2 Lógica e Boas Práticas

- Consumo da API RESTful com Axios.
- Gestão de estado centralizada para autenticação e carrinho de compras.
- Rotas privadas para o painel de administração, protegidas contra acesso não autorizado.
- Componentização e reutilização de elementos de UI para um código limpo e manutenível.
- Validação de formulários no lado do cliente com feedback claro para o utilizador.
- Utilizaçao de tokens nos detalhes dos eventos para protecçao do id

7. BIBLIOTECAS E DEPENDÊNCIAS

7.1 Backend (PHP)

- vlucas/phpdotenv (^5.6): Gestão de variáveis de ambiente.
- PDO: Driver nativo do PHP para acesso a bases de dados.
- Composer: Gestor de dependências e autoloading (PSR-4).

7.2 Frontend (React)

- React (^19.1.1): Biblioteca principal para interfaces.
- React Router DOM (^7.7.1): Gestão de rotas (SPA).
- Axios (^1.11.0): Cliente HTTP para consumo de APIs.
- Bootstrap (^5.3.7): Framework CSS responsivo.
- Styled Components (^6.1.19): CSS-in-JS para styling.
- React Icons (^5.5.0): Biblioteca de ícones.

8. RESULTADO FINAL E FUNCIONALIDADES

8.1 Funcionalidades para Utilizadores

- Registo e autenticação segura.
- Navegação e pesquisa de eventos por categoria.
- Visualização detalhada de eventos.
- Sistema de carrinho de compras funcional.
- Checkout e histórico completo de compras.
- Gestão de perfil pessoal.

8.2 Funcionalidades para Administradores

- Dashboard com estatísticas em tempo real.
- CRUD completo de eventos, utilizadores e categorias.
- Gestão de encomendas e relatórios.
- Sistema de logs de atividade para auditoria.
- Upload e gestão de imagens de eventos.

8.3 Características Técnicas Implementadas

- Arquitetura RESTful: API estruturada com endpoints
- Segurança: Autenticação por sessão, validação de dados, prepared statements, rotecçao com tockets sha256
- Base de dados robusta: Triggers automáticos, views otimizadas, integridade referencial
- Frontend responsivo: Design adaptável a todos os dispositivos
- Gestão de estado: Context API do React para autenticação e carrinho
- Upload de ficheiros: Sistema de upload de imagens para eventos

9. GUIA DE INSTALAÇÃO E CONFIGURAÇÃO

9.1 Pré-requisitos

- XAMPP (Apache + MySQL + PHP 8.0+)
- Node.js 18+ e npm
- Composer
- Git

9.2 Instalação do Backend

- Clonar o projeto para c:\xampp\htdocs\EventHive.
- Navegar para a pasta backend e executar composer install.
- Importar o ficheiro my_script_completed.sql no phpMyAdmin para criar a base de dados eventhive_db.
- Verificar os dados de conexão em backend/config/Database.php.

9.3 Instalação do Frontend

- Navegar para a pasta frontend.
- Executar npm install para instalar as dependências.
- Executar npm start para iniciar o servidor de desenvolvimento.

9.4 URLs do Sistema

- Frontend: http://localhost:3000
- Backend API: http://localhost/EventHive/backend/api/
- Base de dados (phpMyAdmin): http://localhost/phpmyadmin

10. CONCLUSÃO

Este documento serve como referência completa e detalhada para qualquer equipa técnica instalar, manter ou evoluir o EventHive. Todas as decisões, ficheiros, scripts e diagramas estão documentados e referenciados. Para detalhes adicionais, consulte os comentários nos ficheiros de código.

11. BIBLIOGRAFIA / SITES UTILIZADOS

Node.js

Composer

YouTube - Exemplo de Implementação

Hostgator Blog - O que é Composer

PHP: The Right Way

Dev.to - React CRUD com PHP API

PHP.net - JSON Manual

NPM - Node Package Manager

React Bootstrap Docs

Favicon.io - Favicon Converter

Remove.bg - Remove Background